FPT Education

FPT POLYTECHNIC

LẬP TRÌNH ANDROID 2

SERVICE VÀ NOTIFICATION

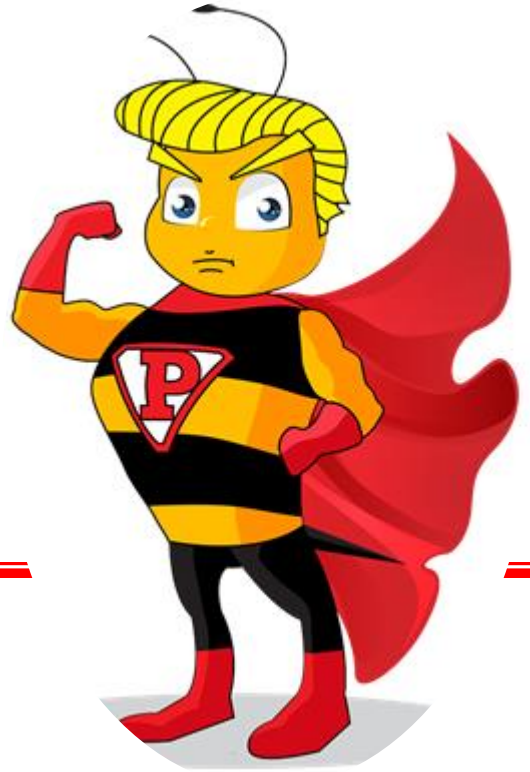
www.poly.edu.vn

GIỚI THIỆU BÀI HỌC

- Tổng quan Service trong Android
- Tạo và sử dụng DrawerNavigation
- Tổng quan Notification trong Android
- Tạo và quản lý Notification

Mục tiêu

- Tổng quan Service trong Android
- Tạo và sử dụng DrawerNavigation
- Tổng quan Notification trong Android
- Tạo và quản lý Notification

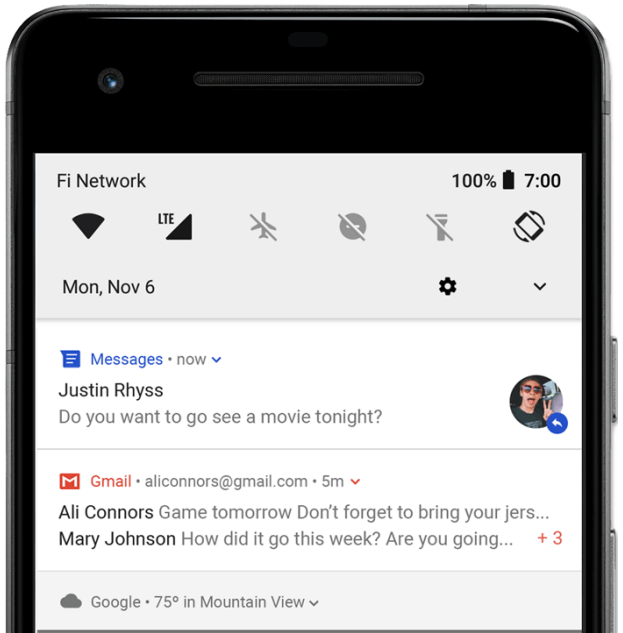


NOTIFICATION

...

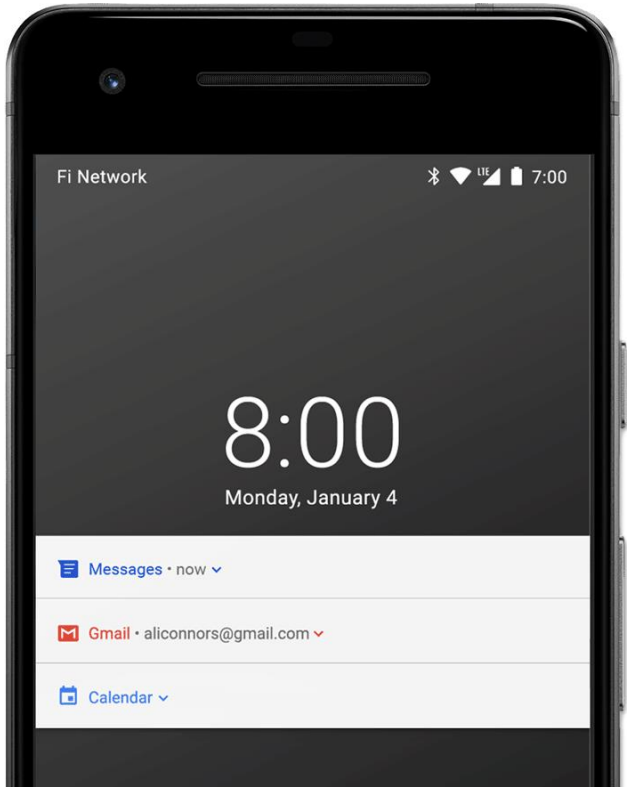
TỔNG QUAN VỀ NOTIFICATION

- **Notificaiton** là một tin nhắn mà Android hiển thị bên ngoài giao diện người dùng của ứng dụng để cung cấp cho người dùng lời nhắc, thông tin liên lạc từ những người khác hoặc thông tin kịp thời khác từ ứng dụng. Người dùng có thể nhấn vào thông báo đó để mở ứng dụng của bạn hoặc thực hiện hành động ngay trong thông báo.

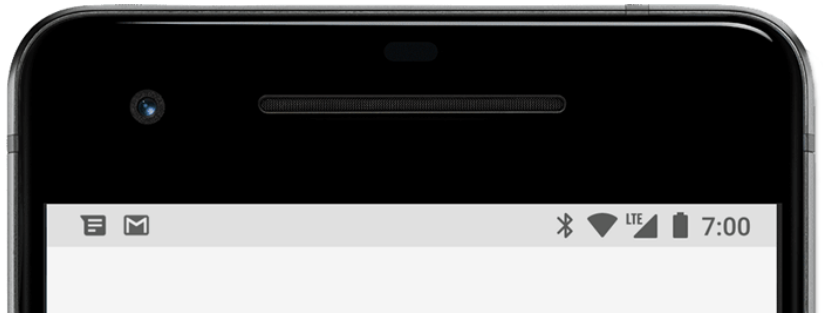


TỔNG QUAN VỀ NOTIFICATION

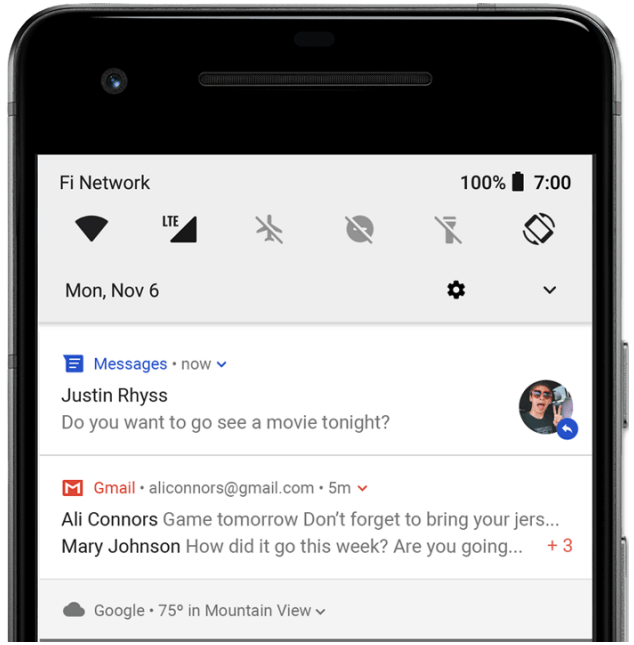
- **Notificaiton** có thể hiển thị cho người dùng ở nhiều vị trí và định dạng khác nhau



- Màn hình khóa



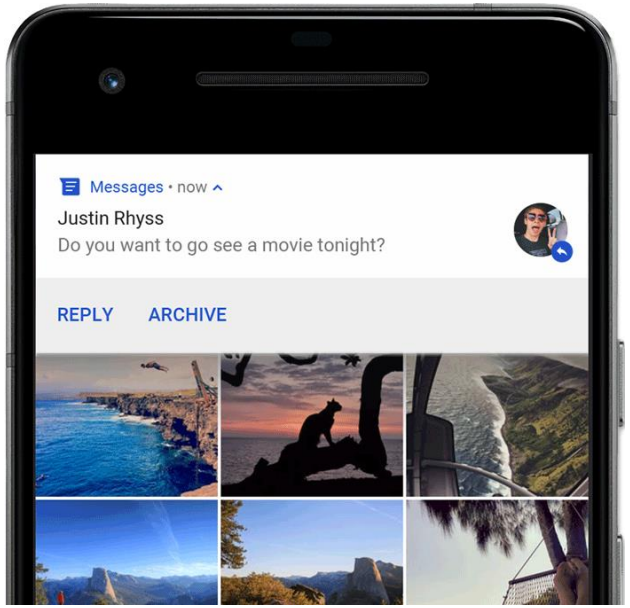
- Statusbar



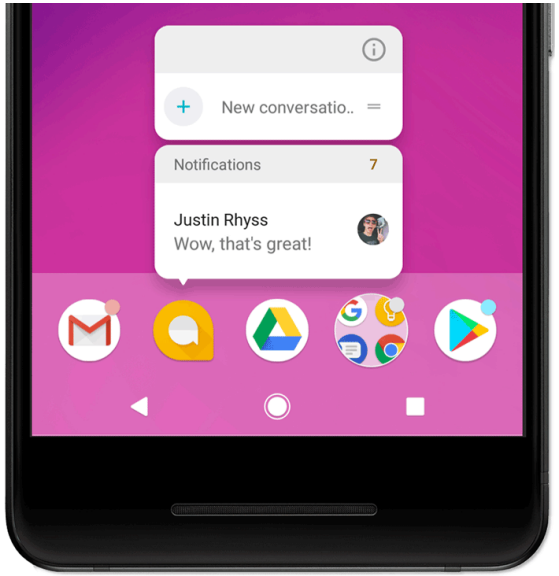
- Trung tâm thông báo / Ngăn xếp thông báo

TỔNG QUAN VỀ NOTIFICATION

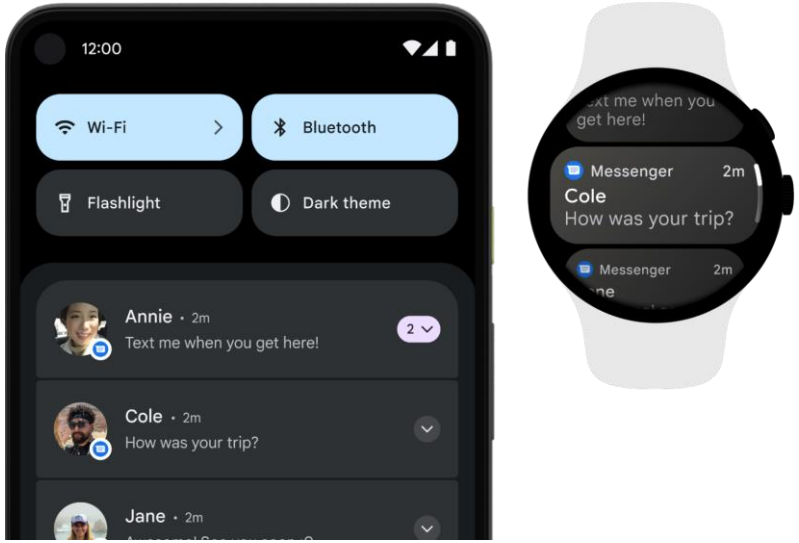
- **Notificaiton** có thể hiển thị cho người dùng ở nhiều vị trí và định dạng khác nhau



- Cửa sổ thông báo quan trọng / Heads-up notification

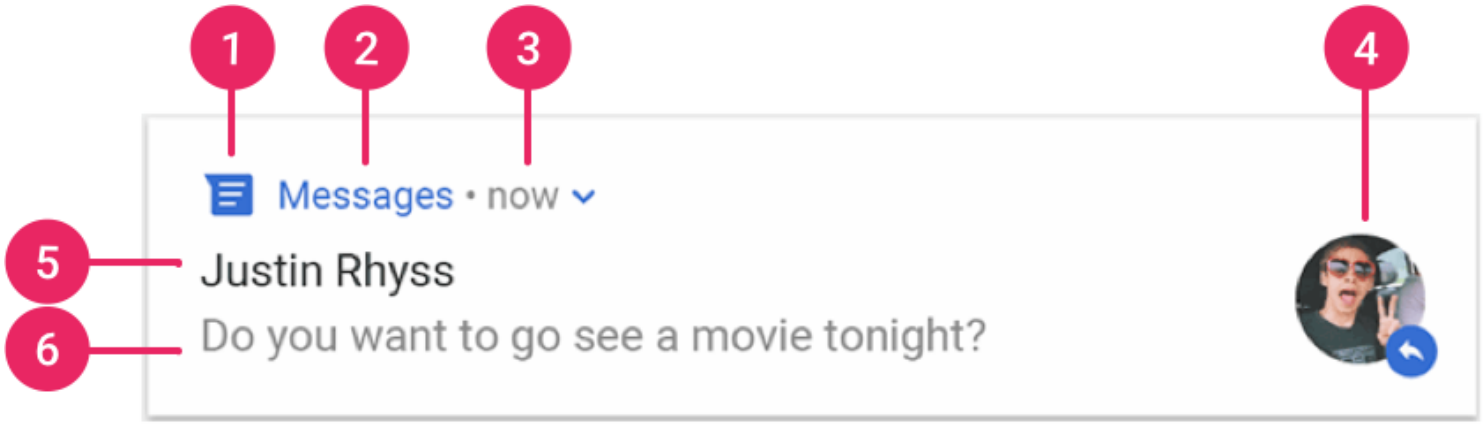


- App icon badge



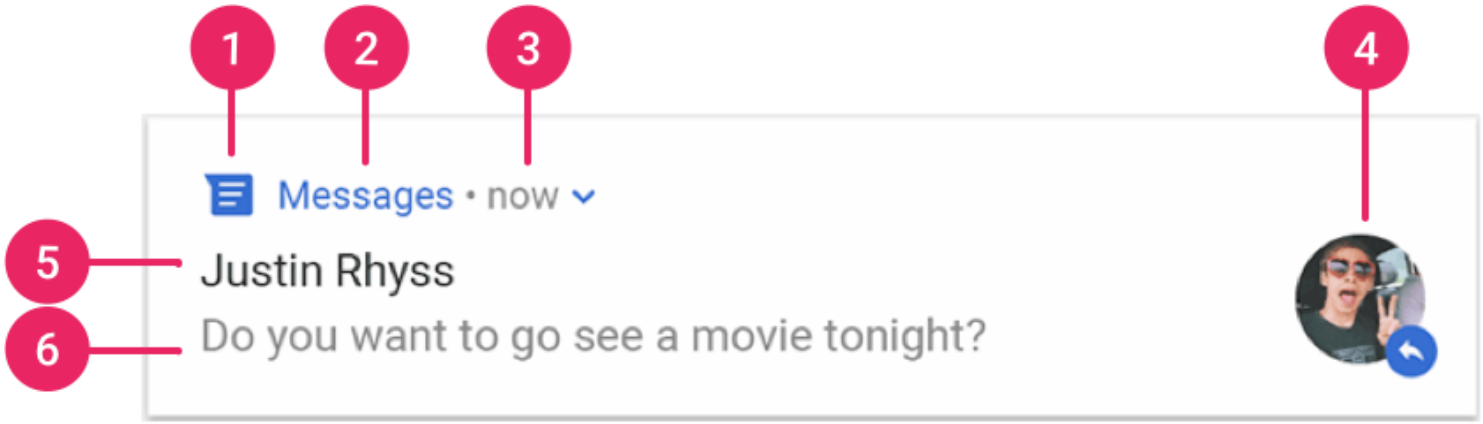
- Thiết bị đeo

THÀNH PHẦN CỦA MỘT NOTIFICATION



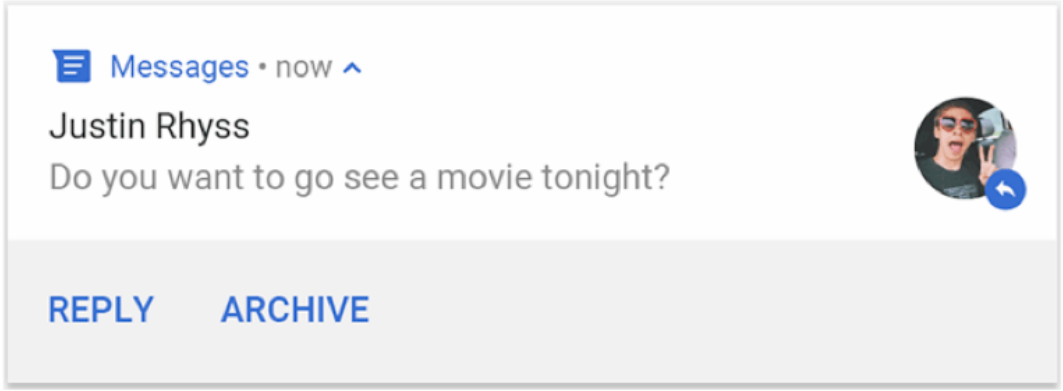
- **(1) Small icon**: Đây là *phần bắt buộc*, sử dụng thuộc tính **setSmallIcon()**
- **(2) Tên ứng dụng**: Do hệ thống cung cấp.
- **(3) Time stamp (Dấu thời gian)**: Thông tin được cung cấp bởi hệ thống, nhưng có thể ghi đè bằng cách sử dụng **setWhen()** hoặc ẩn đi bằng cách sử dụng **setShowWhen(false)**.

THÀNH PHẦN CỦA MỘT NOTIFICATION



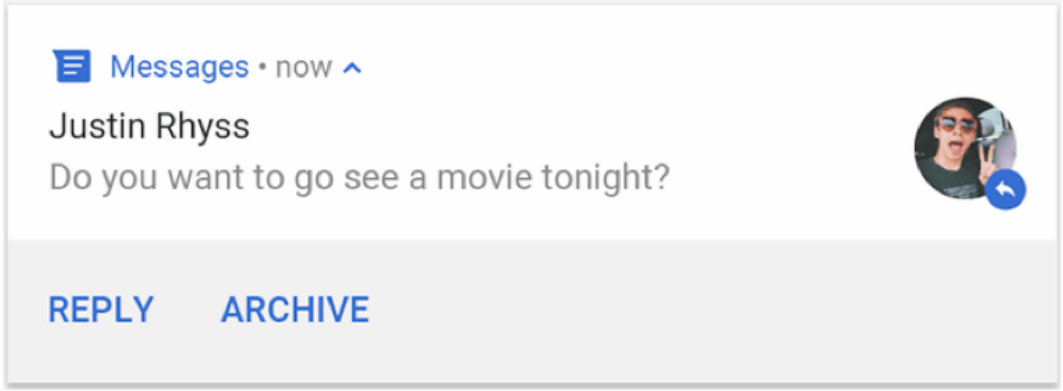
- **(4) Large icon**: Phần này *<u>không bắt buộc</u>* (thường dùng cho ảnh liên hệ, không dùng cho biểu tượng ứng dụng), sử dụng thuộc tính **setLargeIcon()**.
- **(5) Tiêu đề**: Phần này không bắt buộc, sử dụng thuộc tính **setContentTitle()**.
- **(6) Nội dung**: Phần này không bắt buộc, sử dụng thuộc tính **setContentText()**.

HÀNH ĐỘNG TRONG MỘT NOTIFICATION



- Mặc dù không bắt buộc, nhưng khi người dùng nhấn vào thông báo, sẽ hiển thị một số nút thực hiện một số hành động. Ngoài hành động mặc định trong thông báo, ta có thể thêm các nút hành động khác để hoàn tất một nhiệm vụ liên quan đến ứng dụng.

HÀNH ĐỘNG TRONG MỘT NOTIFICATION

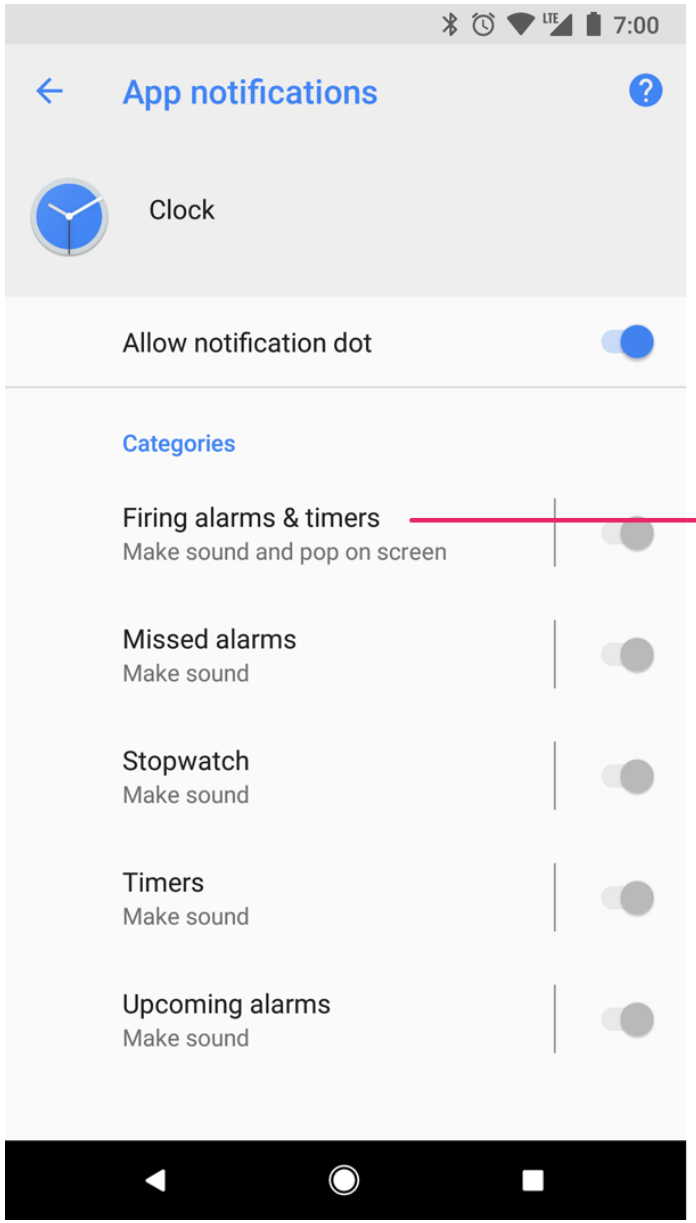


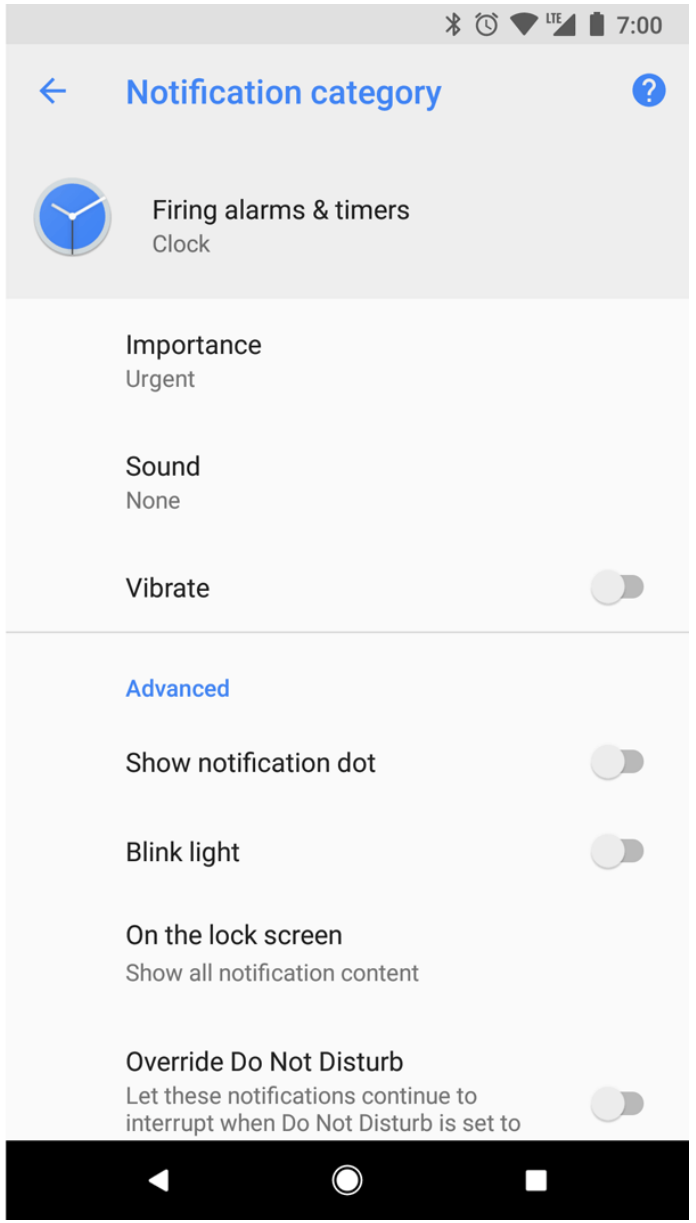
- Kể từ Android 7.0 (API cấp 24), có thể thêm một hành động để trả lời tin nhắn hoặc để nhập văn bản khác ngay trong thông báo.
- Kể từ Android 10 (API cấp 29), hệ thống có thể tự động tạo các nút hành động bằng các hành động dựa trên ý định đề xuất.

NOTIFICATION CHANNELS

- Từ Android 8.0 (API cấp 26), ta phải chỉ định channels cho tất cả thông báo thì thông báo mới xuất hiện. Bằng cách phân loại thông báo vào các channels, người dùng có thể tắt các channels thông báo cụ thể của ứng dụng (thay vì tắt tất cả các thông báo), đồng thời có thể kiểm soát các tuỳ chọn hình ảnh và âm thanh cho từng channels. Ngoài ra, người dùng còn có thể nhấn và giữ một thông báo để thay đổi cách hoạt động của channels liên kết.

NOTIFICATION CHANNELS





- Các chế độ cài đặt thông báo của ứng dụng Đồng hồ và một trong các channels thông báo của ứng dụng đó

NOTIFICATION CHANNELS

- Một ứng dụng có thể có nhiều channels thông báo, mỗi loại thông báo mà ứng dụng tạo ra sẽ có một channels riêng. Ứng dụng cũng có thể tạo các channels thông báo để phản hồi các lựa chọn của người dùng ứng dụng.
- ***Ví dụ:*** Ta có thể thiết lập các channels riêng cho từng nhóm cuộc trò chuyện do người dùng tạo trong ứng dụng nhắn tin.
- Channels cũng là nơi chỉ định mức độ quan trọng của thông báo (từ Android 8.0 trở lên). Vì vậy, tất cả các thông báo được đưa lên cùng một channels đều có cùng cách hoạt động.

TẠO MỘT NOTIFICATION

Bước 1: Tạo file config cho notification

```
public class ConfigNotification extends Application {
    2 usages
    public static final String CHANNEL_ID = "FPTPOLYTECHNIC";

    @Override
    public void onCreate() {
        super.onCreate();
        config();
    }

    1 usage
    private void config(){
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            //Tên của Notification Channel cần đăng ký
            CharSequence name = getString(R.string.channel_name);
            //Mô tả của Notification Channel
            String description = getString(R.string.channel_description);
            //Độ ưu tiên của Notification
            int importance = NotificationManager.IMPORTANCE_DEFAULT;
            //Sử dụng RingtoneManager để lấy uri của âm thanh notification theo máy
            Uri uri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_RINGTONE);
            //Tạo thêm một audioAttributes
            AudioAttributes audioAttributes = new AudioAttributes.Builder()
                    .setUsage(AudioAttributes.USAGE_NOTIFICATION)
                    .build();
            //Đăng ký NotificationChannel
            NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name, importance);
            channel.setDescription(description);
            //Set âm thanh cho notifcation
            channel.setSound(uri, audioAttributes);
            //Đăng ký channel với hệ thống
            NotificationManager notificationManager = getSystemService(NotificationManager.class);
            notificationManager.createNotificationChannel(channel);
        }
    }
}
```

TẠO MỘT NOTIFICATION

- **Bước 2:** Trong Android Manifest khai báo quyền Post Notification cho ứng dụng

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
```

- **Bước 3:** Khai báo thêm thuộc tính name trong thẻ <application>

```
<application
    android:name=".ConfigNotification"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
```

TẠO MỘT NOTIFICATION

Bước 4: Tạo một notification

```
//Ta sẽ dùng bitmap để đưa hình vào notification
Bitmap logo = BitmapFactory.decodeResource(getResources(), R.mipmap.logofpt);

NotificationCompat.Builder builder = new NotificationCompat.Builder( context: this, ConfigNotification.CHANNEL_ID)
        //icon sẽ hiện thị trên status bar
        .setSmallIcon(R.mipmap.logofpt)
        //Tiêu đề của notification
        .setContentTitle("Chào mừng đến với FPT Polytechnic")
        //Nội dung của notification
        .setContentText("Android 2")
        //Truyền một hình ảnh vào notification
        .setStyle(new NotificationCompat.BigPictureStyle()
                .bigPicture(logo)
                .bigLargeIcon( b: null)
        )
        //Hiển thị icon bên phải khi notification ở dạng thu gọn
        .setLargeIcon(logo)
        .setColor(Color.RED)
        .setAutoCancel(true);
NotificationManagerCompat notificationManager = NotificationManagerCompat.from( context: this);
//Code kiểm tra quyền notification trên thiết bị
if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.POST_NOTIFICATIONS)
        == PackageManager.PERMISSION_GRANTED) {
    //Nếu đã có quyền, ta thực hiện push notification
    //mỗi thông báo được push cần có 1 id riêng, sử dụng Date().getTime() để tạo ID cho thông báo
    notificationManager.notify((int) new Date().getTime(), builder.build());

} else {
    //Nếu ko có quyền, thì sẽ thực hiện xin quyền
    ActivityCompat.requestPermissions( activity: this,
            new String[]{Manifest.permission.POST_NOTIFICATIONS}, requestCode: 7979);
}
```

Bước 5: Tạo hàm xin quyền nếu ứng dụng chưa được cấp quyền thông báo

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == 7979) {
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                sendNotification();
            }
        }
    }
}
```

Fpt University
FPT POLYTECHNIC

SERVICE

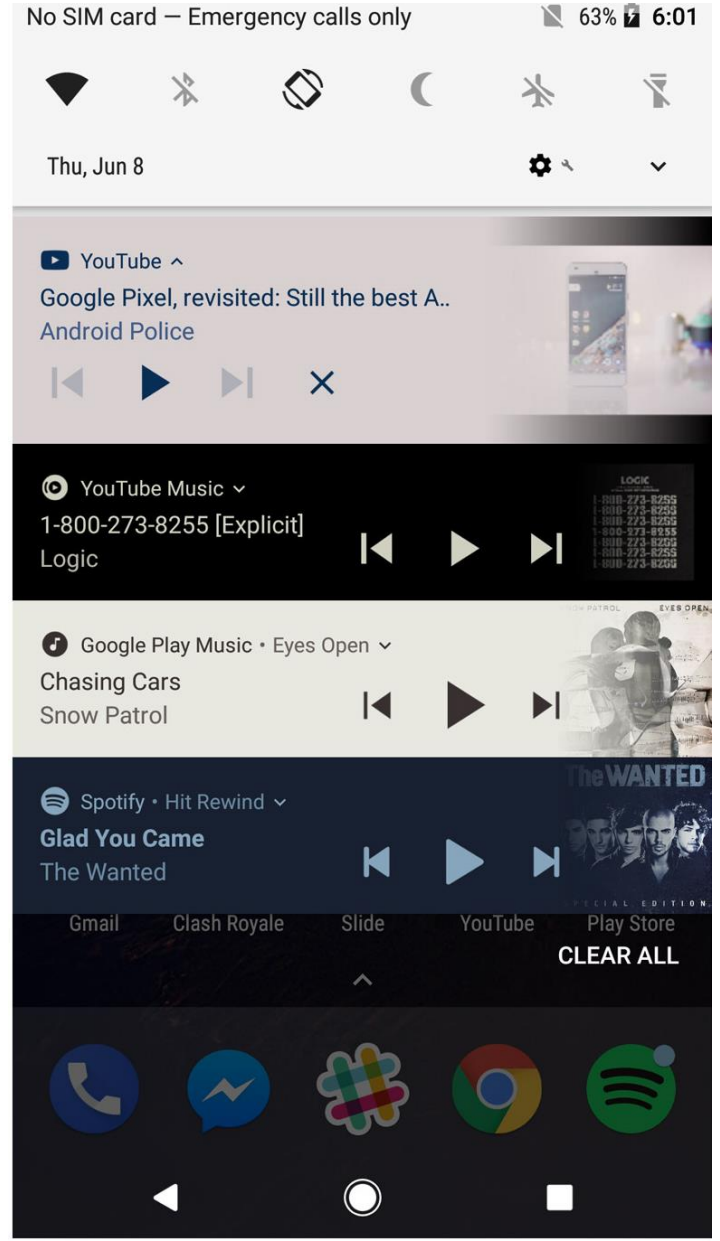
…

TỔNG QUAN VỀ SERVICE

- Service là thành phần ứng dụng có thể thực hiện các thao tác đòi hỏi tốn nhiều thời gian và tài nguyên
- Service không cung cấp giao diện cho người dùng
- Service chạy ngầm để thực hiện các công việc như nghe nhạc, thực hiện ghi và đọc file, hoặc tương tác với Content Provider

FOREGROUND SERVICE

- Là Service mà người dùng biết là Service đang chạy và hệ thống sẽ không Kill Service khi bộ nhớ xuống thấp
- Phải cung cấp một Notification trên Status bar thể hiện Service đang chạy trừ khi Service bị dừng hoặc bị hủy từ Foreground

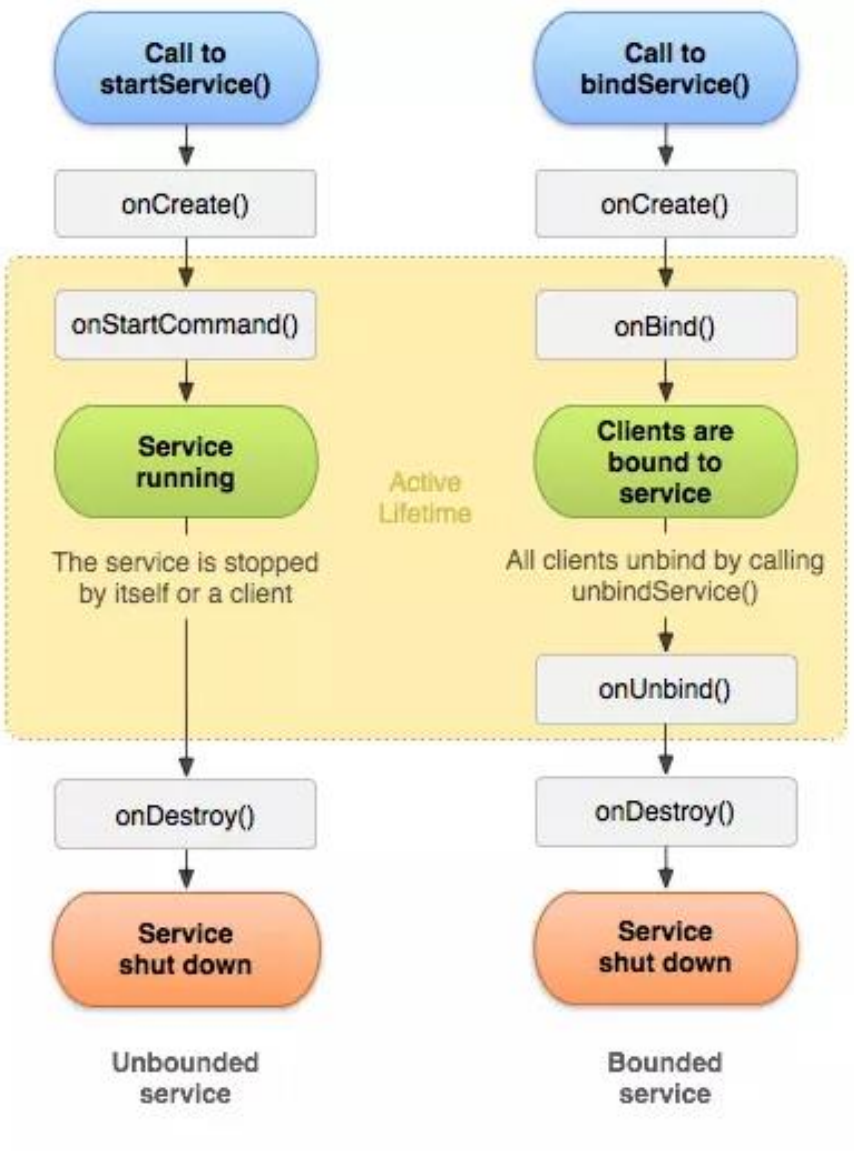


BACKGROUND SERVICE

- Là Service chạy mà người dùng không ý thức được là Service đang chạy và không có tương tác với người dùng

- Ví dụ:
  - nhận tin nhắn đến, nhận cuộc gọi đến, nhận email,...
  - Đặt báo thức vào lúc 07:44 AM, khi đồng hồ hệ thống điểm 07:44 AM, thiết bị sẽ thông báo báo thức cho người dùng bằng âm thanh và notification.

VÒNG ĐỜI SERVICE



- Service đã khởi tạo (**Started Service**) là service được khởi tạo khi một thành phần ứng dụng (ví dụ như Activity) khởi tạo Service bằng lời gọi phương thức **startService()**
- Ngay khi khởi tạo xong, Service có thể chạy nền vô hạn kể cả khi thành phần khởi tạo Service (ví dụ như Activity khởi tạo Service này) đã bị hủy
- Thông thường, Service đã khởi tạo thực hiện một thao tác đơn giản và không trả lại một kết quả cụ thể.
- ***Ví dụ:*** Download và Upload một file trên mạng. Khi Service hoàn thành nhiệm vụ việc download và upload, Service sẽ tự động dừng lại.

Bound Service

- Service ràng buộc (**Bound Service**) khi một thành phần ứng dụng ràng buộc Service bằng lời gọi **bindService()**
- Service ràng buộc có giao diện client-server cho phép các thành phần ứng dụng tương tác với Service, gửi yêu cầu, nhận kết quả.
- Một Service ràng buộc sẽ chạy cho đến khi vẫn còn thành phần ứng dụng ràng buộc với Service.
- Nhiều thành phần có thể cùng ràng buộc Service, khi tất cả thành phần không ràng buộc Service nữa, Service sẽ bị hủy

- Một Service có thể hoạt động theo cả 2 cách: vừa được khởi tạo (chạy vô hạn) và vừa cho phép ràng buộc (binding). Khi đó, bạn chỉ cần miêu tả 2 phương thức **onStartCommand()** cho phép thành phần khởi tạo nó và onBind() cho phép ràng buộc nó.
- Dù ứng dụng của bạn có được khởi tạo, ràng buộc hoặc vừa khởi tạo và ràng buộc, bất kỳ thành phần ứng dụng nào cũng có thể sử dụng Service (thậm chí từ một ứng dụng khác) giống như cách mà bất kỳ thành phần nào cũng có thể sử dụng một acvitity bằng cách khởi tạo nó sử dụng Intent
- Tuy nhiên bạn có thể khai báo Service là private trong file **AndroidManifest.xml** (khai báo **android:exported="false"** cho Service trong AndroidManifest)

PHƯƠNG THỨC CALLBACK CỦA SERVICE

❑ onStartCommand()

- ❖ Hệ thống sẽ gọi phương thức này khi một component khác như activity yêu cầu service bắt đầu bằng cách gọi **startService()**. Khi phương thức này được thực thi, service sẽ khởi động và chạy vô thời hạn trong nền. Nếu chúng ta thực hiện điều này trong code thì phải dừng service sau khi hoàn công việc bằng cách gọi phương thức **stopSelf()** hoặc **stopService()**.

PHƯƠNG THỨC CALLBACK CỦA SERVICE

onStartCommand()

- Trong Android, phương thức onStartCommand() phải trả về một kiểu Interger, đây là giá trị mô tả cách hệ thống sẽ tiếp tục service trong trường hợp hệ thống hủy nó.
- Phương thức onStartCommand() sẽ trả về một giá trị là một trong những hằng số sau: **START_REDELIVER_INTENT**, **START_STICKY**, **START_NOT_STICKY**

PHƯƠNG THỨC CALLBACK CỦA SERVICE

Tùy chọn	Mô tả
START_STICKY	Nó sẽ **khởi động** lại service trong trường hợp dữ liệu truyền cho phương thức **onStartCommand()** là **NULL**.
START_NOT_STICKY	Nó sẽ **không khởi động** lại service và nó hữu ích cho những service chạy định kỳ. Service sẽ khởi động lại khi có lệnh **startService()**. Đó là cách tốt nhất để tránh service chạy khi không cần thiết.
START_REDELIVER_INTENT	Nó giống như **START_STICKY** và nó tạo lại service, gọi **onStartCommand()** với intent cuối cùng đã được gửi đến service.

PHƯƠNG THỨC CALLBACK CỦA SERVICE

onBind()

- Hệ thống sẽ gọi phương thức này khi một component khác muốn liên kết với service bằng cách gọi **bindService()**. Trong quá trình thực hiện phương thức này, chúng ta phải cung cấp giao diện cho khách hàng để liên lạc với service bằng cách trả về một đối tượng **IBinder**.

PHƯƠNG THỨC CALLBACK CỦA SERVICE

❑ onCreate()

- ❖ Hệ thống sẽ gọi phương thức này khi Service được tạo bằng cách sử dụng phương thức **onStartCommand()** hoặc **onBind()**. Trong trường hợp nếu service đã chạy, thì phương thức này sẽ không được gọi.

PHƯƠNG THỨC CALLBACK CỦA SERVICE

❑ onDestroy()

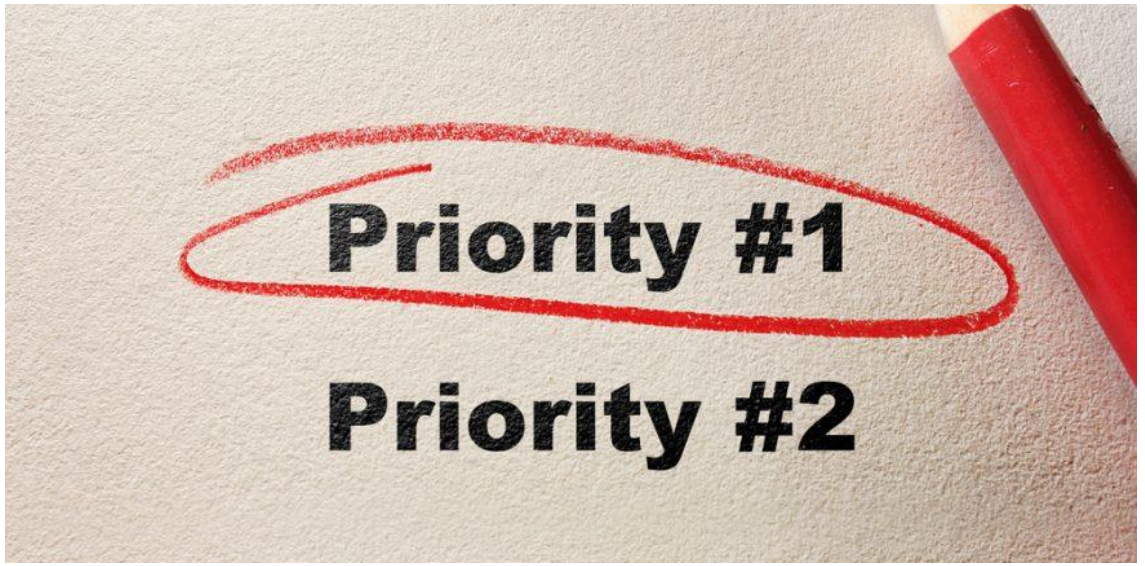
- ❖ Hệ thống sẽ gọi phương thức này khi service không còn được sử dụng và đang bị hủy. Đây là phương thức cuối cùng mà service sẽ nhận được và chúng ta cần triển khai phương thức này trong service của mình để dọn sạch mọi tài nguyên không sử dụng như **threads**, **receivers** hoặc **listeners**.

PHƯƠNG THỨC CALLBACK CỦA SERVICE

- Tóm lại, nếu bắt đầu service bằng phương thức gọi **startService()** thì service sẽ chạy liên tục ngay cả khi component khởi động service bị hủy, cho đến khi dừng service bằng cách sử dụng **stopService()** hoặc nó tự dừng với phương thức **stopSelf()**.
- Tương tự, nếu tạo một service bằng phương thức gọi **bindService()**, service sẽ chạy miễn là component đó bị ràng buộc với nó. Sau khi service không bị ràng buộc từ tất cả các client, hệ thống sẽ hủy nó.
- Trong Android, vòng đời của service là có một tập hợp các phương thức callback cần được triển khai để theo dõi trạng thái service và thực hiện những điều cần thiết trong thời gian thích hợp.

ĐỘ ƯU TIÊN CÁC LOẠI SERVICE

- Android bắt buộc phải dừng một service khi bộ nhớ ít và phải khôi phục tài nguyên hệ thống cho Activity đang được sử dụng. Nếu Service được ràng buộc với một Activity đang sử dụng thì rất ít khả năng bị tiêu hủy. Tuy nhiên, nếu Service được khai báo và chạy ở chế độ Foreground thì nó cũng khó để xóa bớt.

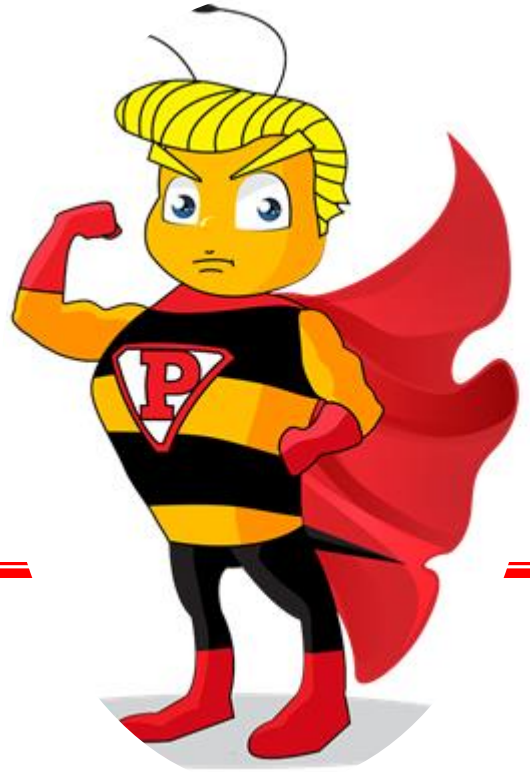


ĐỘ ƯU TIÊN CÁC LOẠI SERVICE

- Về trường hợp Service là Started và chạy lâu dài, hệ thống sẽ làm giảm vị trí ưu tiên của nó, bởi vì phụ thuộc vào process thì các loại service sẽ được xếp theo độ ưu tiên sau: **Bound Service** khó bị tiêu hủy nhất, tiếp theo là **Foreground Service** và **Background Service**.

- Dựa vào những lí do trên, **Background Service** được coi là Service dễ bị tiêu hủy nhất nên cần phải xử lý một cách thích hợp, tùy thuộc vào giá trị trả về trong **onStartCommand()** mà Service có thể được khởi động lại.

Fpt University
FPT POLYTECHNIC



WORKMANAGER

...

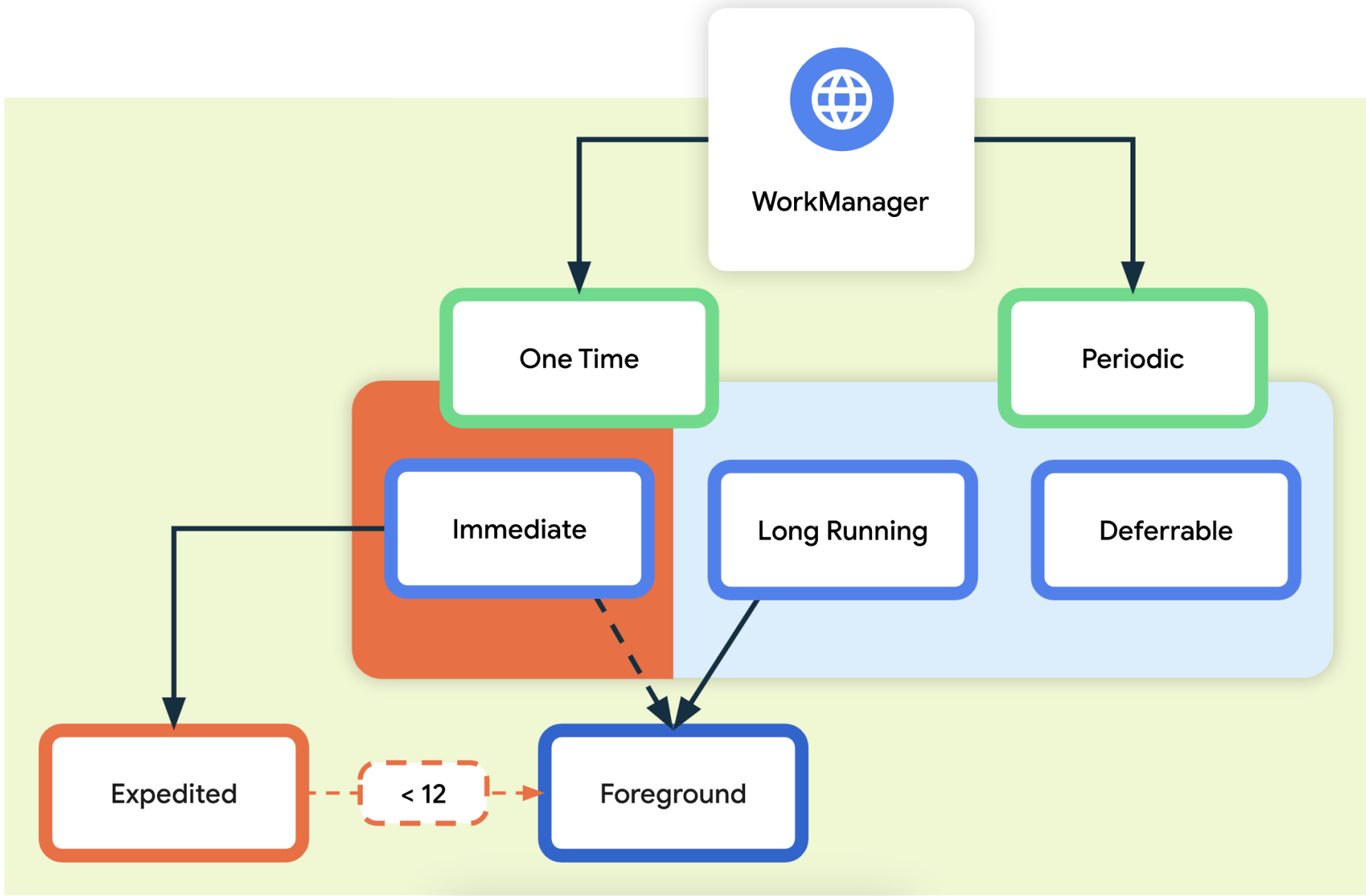
TỔNG QUAN WORKMANAGER

- **WorkManager** là giải pháp được đề xuất cho công việc liên tục. Khả năng làm việc liên tục khi được lên lịch thông qua các lần khởi động lại ứng dụng và khởi động lại hệ thống. Vì hầu hết quá trình xử lý nền đều diễn ra hiệu quả nhất thông qua khả năng làm việc liên tục, **WorkManager** là API chính nên dùng để xử lý nền.

CÁC LOẠI CÔNG VIỆC LIÊN TỤC

- WorkManager xử lý ba loại công việc liên tục:
  - **Ngay lập tức (Immediate)**: Những tác vụ phải bắt đầu ngay và hoàn thành sớm. Có thể được ưu tiên.
  - **Lâu dài (Long Running)**: Những tác vụ có thể chạy lâu hơn và có thể kéo dài hơn 10 phút.
  - **Có thể trì hoãn (Deferrable)**: Những tác vụ đã lên lịch bắt đầu sau và có thể chạy định kỳ.

CÁC LOẠI CÔNG VIỆC LIÊN TỤC



Fpt University
FPT POLYTECHNIC


FPT Education
FPT POLYTECHNIC
Thank you
TIN HỌC VĂN PHÒNG QUỐC TẾ THEO CHUẨN ICDL
Syllabus 5 - Office 2007
LUẬT KINH TẾ
KỸ THUẬT SOẠN THẢO VÀ TRÌNH BÀY VĂN BẢN